

# Datenbanken 2

## Physische Datenorganisation

Nikolaus Augsten

`nikolaus.augsten@plus.ac.at`

FB Informatik

Universität Salzburg



database  
research group

<https://dbresearch.uni-salzburg.at>

WS 2024/25

Version 1. Oktober 2024

# Inhalt

- 1 Speichermedien
- 2 Speicherzugriff
- 3 Dateien, Datensätze, Blöcke

# Inhalt

- 1 Speichermedien
- 2 Speicherzugriff
- 3 Dateien, Datensätze, Blöcke

# Speichermedien/1

- **Verschiedene Arten** von Speichermedien sind für Datenbanksysteme relevant.
- **Klassifizierung** der Speichermedien nach:
  - Zugriffsgeschwindigkeit
  - Kosten pro Dateneinheit
  - Flüchtige vs. persistente Speicher
    - Flüchtig (volatile): Inhalt geht nach Ausschalten verloren
    - Persistent (non-volatile): Inhalt bleibt auch nach Ausschalten
  - **Verlässlichkeit**
    - Physische Fehler des Speichermediums
    - Datenverlust durch Stromausfall oder Systemabsturz

# Speichermedien/2

- Cache
  - flüchtig
  - am schnellsten und am teuersten
  - von Prozessor verwaltet
  - 3 Levels: L1 (z.B. 128KB bei Athlon 64), L2, L3
  
- Hauptspeicher (RAM)
  - flüchtig
  - schneller Zugriff (unter 100 ns;  $1 \text{ ns} = 10^{-9} \text{ s}$ )
  - Datentransferrate z.B. 25,6 GB/s (DDR4-3200)
  - meist zu klein (oder zu teuer) um gesamte Datenbank zu speichern
    - mehrere GB weit verbreitet
    - Preise derzeit ca. 4'000 EUR/TB (DRAM)

# Speichermedien/3

- Flash memory (SSD - Solid State Drive)
  - persistent
  - Speichermedium: NAND Flash Technologie (Firmware: auch NOR)
  - Zugriffsmuster:
    - Random-access: Daten können in beliebiger Reihenfolge gelesen werden
    - Block-basiert: Granularität nicht Bytes sondern Seiten (z.B. à 4096 B)
    - Puffer-basiert: Daten müssen für Zugriff in RAM geladen werden
  - Lesezugriff langsamer als RAM ( $x0$  bis  $x00 \mu s$ ;  $1 \mu s = 10^{-6}s$ )
  - schnellster persistenter Speicher
  - hohe sequentielle Datentransferrate (4-5 GB/s schreiben, 6-7 GB/s lesen für NVMe)
  - größer und billiger als Hauptspeicher
  - Preise derzeit ca. 60-70 EUR/TB
  - auch als EEPROM bekannt (Electrically Erasable Programmable Read-Only Memory)

# Speichermedien/4

- Festplatte

- persistent
- Daten sind auf Magnetscheiben gespeichert, mechanische Drehung
- Zugriffsmuster: wie SSD
- sehr viel langsamer als RAM (Zugriff im ms-Bereich;  $1 \text{ ms} = 10^{-3} \text{ s}$ )
- sequentielles Lesen: 25–100 MB/s
- billig: Preise teils unter 25 EUR/TB
- sehr viel mehr Platz als im Hauptspeicher: mehrere TB
- Kapazitäten stark ansteigend (Faktor 2 bis 3 alle 2 Jahre)
- Hauptmedium für Langzeitspeicher: speichert gesamte Datenbank
- Diskette vs. Festplatte

# Speichermedien/5

- Optische Datenträger

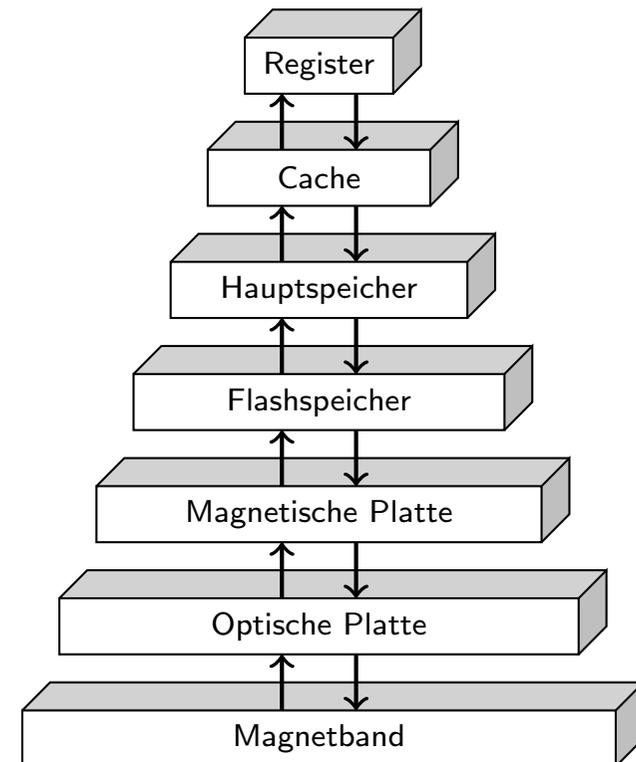
- persistent
- Daten werden optisch via Laser von einer drehenden Platte gelesen
- lesen und schreiben langsamer als auf magnetischen Platten
- sequentielles Lesen: 1 Mbit/s (CD) bis 400 Mbit/s (Blu-ray)
- verschiedene Typen:
  - CD-ROM (640-700 MB), DVD (4.7-17 GB), Blu-ray (25-129 GB)
  - write-once, read-many (WORM) oder mehrfach schreibbar (CD-RW, DVD-RW, DVD-RAM)
  - nicht für langfristige Speicherung (WORM max. 5-10 Jahre)
  - M-Disc (Millennial Disk) für Archivierung (angeblich 1000 Jahre)

# Speichermedien/6

- Band
  - persistent
  - Zugriff sehr langsam, da sequentieller Zugriff
  - Datentransfer jedoch z.T. wie Festplatte (bis zu 400 MB/s, komprimiert noch mehr)
  - sehr hohe Kapazität (mehrere TB)
  - sehr billig (ab 10 EUR/TB)
  - hauptsächlich für Backups genutzt
  - Band kann aus dem Laufwerk genommen werden
  - Tape Library für sehr große Datenmengen
    - Anschaffung teuer
    - bis x00 Petabyte (1 petabyte =  $10^{15}$  bytes)

# Speichermedien/7

- Speichermedien können hierarchisch nach Geschwindigkeit und Kosten geordnet werden:
- **Primärspeicher:** flüchtig, schnell, teuer
  - z. B. Cache, Hauptspeicher
- **Sekundärspeicher:** persistent, langsamer und günstiger als Primärspeicher
  - z. B. Magnetplatten, Flash Speicher
  - auch Online-Speicher genannt
- **Tertiärspeicher:** persistent, sehr langsam, sehr günstig
  - z. B. Magnetbänder, optischer Speicher
  - auch Offline-Speicher genannt
- Datenbank muss mit Speichermedien auf allen Ebenen umgehen

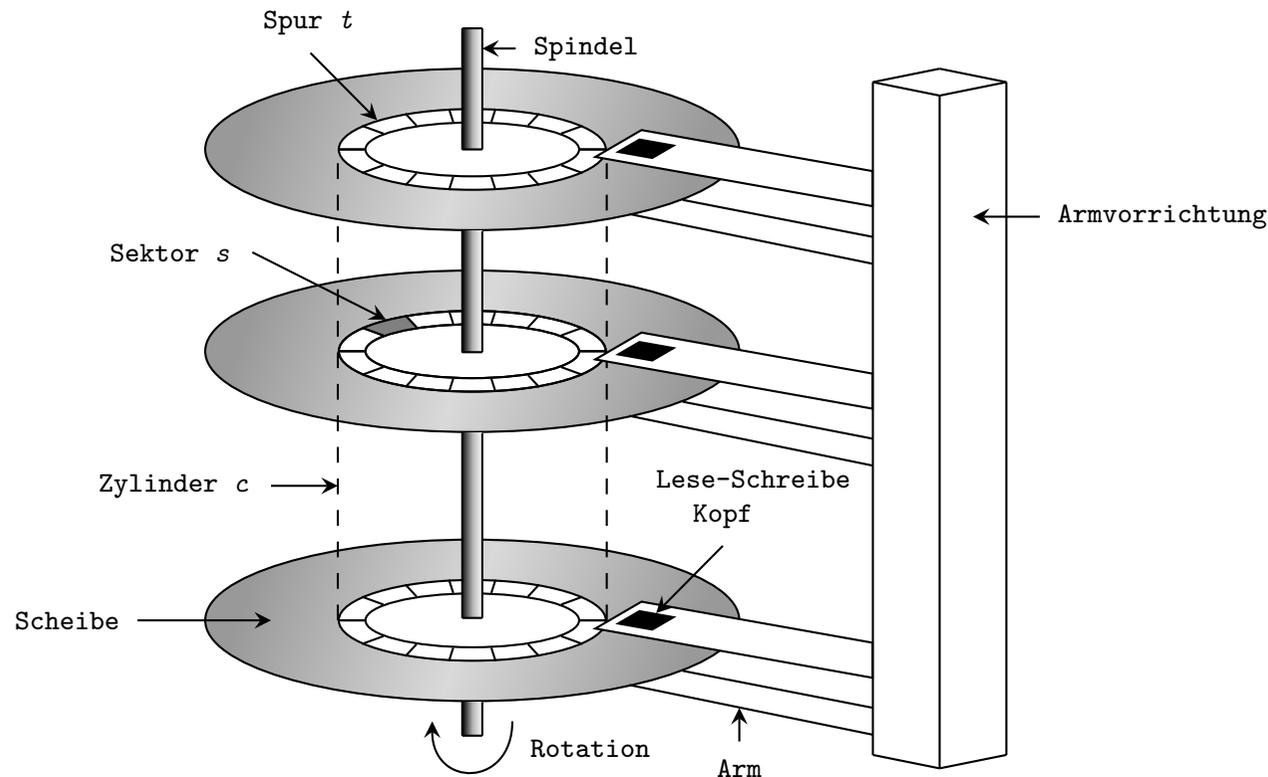


# Aktuelle Entwicklung: Non-Volatile Memory - NVRAM

- **Ziel:** Lücke zwischen RAM und SSD/Festplatte füllen
  - verwendbar wie RAM (Byte-Adressierung und geringe Latenz)
  - jedoch persistent (non-volatile)
  - günstiger als RAM
  - verschiedene Technologien werden erforscht
- **3D XPoint** Technologie (Intel & Micron) seit 2017 auf dem Markt
  - als Speichermodul wie RAM (Intel Optane Memory, 128/256/512 GB Module)
  - auf SSD-Laufwerken verbaut (Intel Optane SSD)
- **Geschwindigkeit** zwischen RAM und SSD (z.B. Latenz ca. 300 ns)
- sehr interessant für **Datenbanksysteme**:
  - speichern von Log
  - Hybrid-Speicher in Kombination mit SSD oder Festplatte
  - Ersatz für langsamere, persistente Speicher

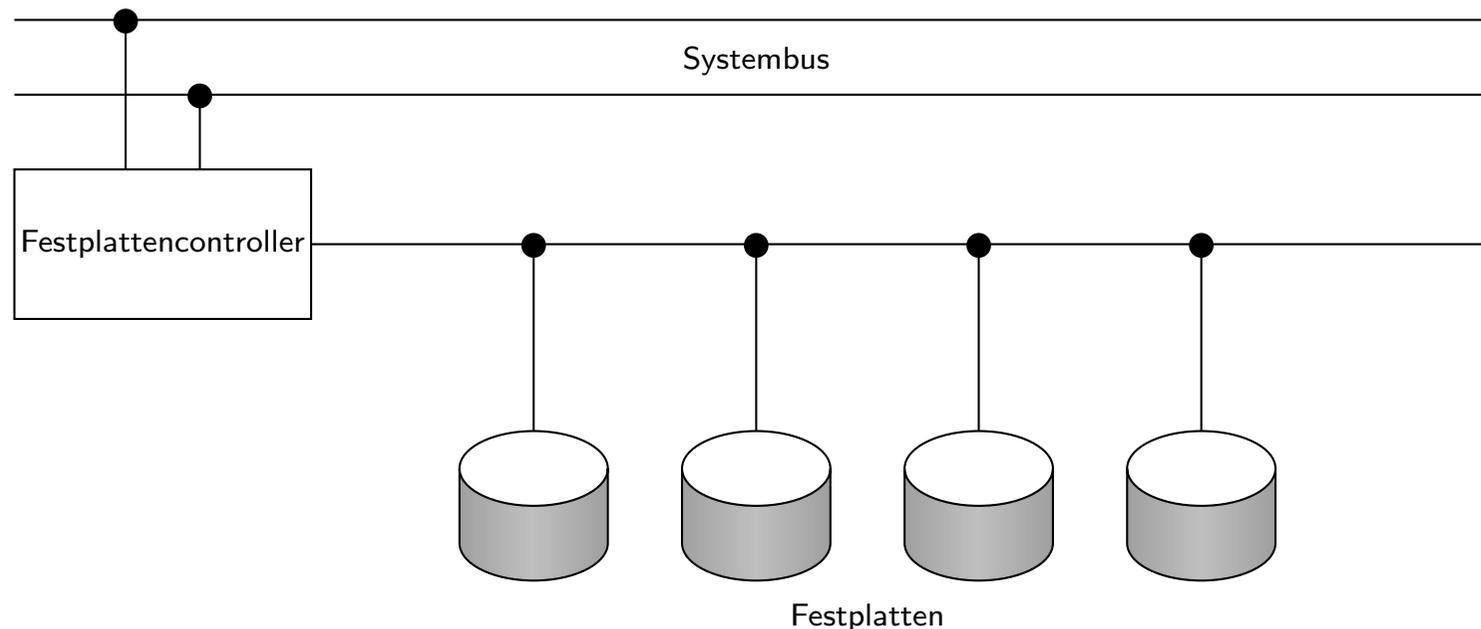
# Festplatten/1

- Meist sind Datenbanken auf magnetischen Platten gespeichert, weil:
  - die Datenbank zu groß für den Hauptspeicher ist
  - der Plattenspeicher persistent ist
  - Plattenspeicher billiger als Hauptspeicher ist
- Schematischer Aufbau einer Festplatte:



# Festplatten/2

- **Controller:** Schnittstelle zwischen Computersystem und Festplatten:
  - übersetzt high-level Befehle (z.B. bestimmten Sektor lesen) in Hardware Aktivitäten (z.B. Disk Arm bewegen und Sektor lesen)
  - für jeden Sektor wird Checksum geschrieben
  - beim Lesen wird Checksum überprüft



# Festplatten/3

Drei Arbeitsvorgänge für Zugriff auf Festplatte:

- **Spurwechsel** (seek time): Schreib-/Lesekopf auf richtige Spur bewegen
- **Latenz** (rotational latency): Warten bis sich der erste gesuchte Sektor unter dem Kopf vorbeibewegt
- **Lesezeit**: Sektoren lesen/schreiben, hängt mit Datenrate (data transfer rate) zusammen

$$\text{Zugriffszeit} = \text{Spurwechsel} + \text{Latenz} + \text{Lesezeit}$$

# Festplatten/4

## Performance Parameter von Festplatten

- **Spurwechsel:** gerechnet wird mit mittlerer Seek Time (=1/2 Worst Case Seek Time)
- **Latenz:**
  - errechnet sich aus Drehzahl (5400rpm-15000rpm)
  - rpm = revolutions per minute
  - maximale Latenz [s] =  $60 / \text{Drehzahl [rpm]}$
  - mittlere Latenz: 1/2 maximale Latenz (2ms-5.5ms)
- **Datenrate:** Rate mit der Daten gelesen/geschrieben werden können (z.B. 25-100 MB/s)
- **Mean time to failure (MTTF):** mittlere Laufzeit bis zum ersten Mal ein Hardware-Fehler auftritt
  - typisch: mehrere Jahre
  - keine Garantie, nur statistische Wahrscheinlichkeit

# Festplatten/5

- **Block:** (auch “Seite”) zusammenhängende Reihe von Sektoren auf einer bestimmten Spur
- **Interblock Gaps:** ungenützter Speicherplatz zwischen Sektoren
- ein Block ist eine **logische Einheit** für den Zugriff auf Daten.
  - Daten zwischen Platte und Hauptspeicher werden in Blocks übertragen
  - Datenbank-Dateien sind in Blocks unterteilt
  - Block Größen: 4-16 kB
    - kleine Blocks: mehr Zugriffe erforderlich
    - große Blocks: Ineffizienz durch nur teilweise gefüllte Blocks

# Integrierte Übung 1.1

Betrachte folgende Festplatte: Sektor-Größe  $B = 512$  Bytes, Sektoren/Spur  $S = 20$ , Spuren pro Scheibenseite  $T = 400$ , Anzahl der beidseitig beschriebenen Scheiben  $D = 15$ , mittlerer Spurwechsel  $sp = 30ms$ , Drehzahl  $dz = 2400rpm$  (Interblock Gaps werden vernachlässigt).

Bestimme die folgenden Werte:

- a) Kapazität der Festplatte
- b) mittlere Zugriffszeit (1 Sektor lesen)

# SSD - Solid State Drive - Aufbau

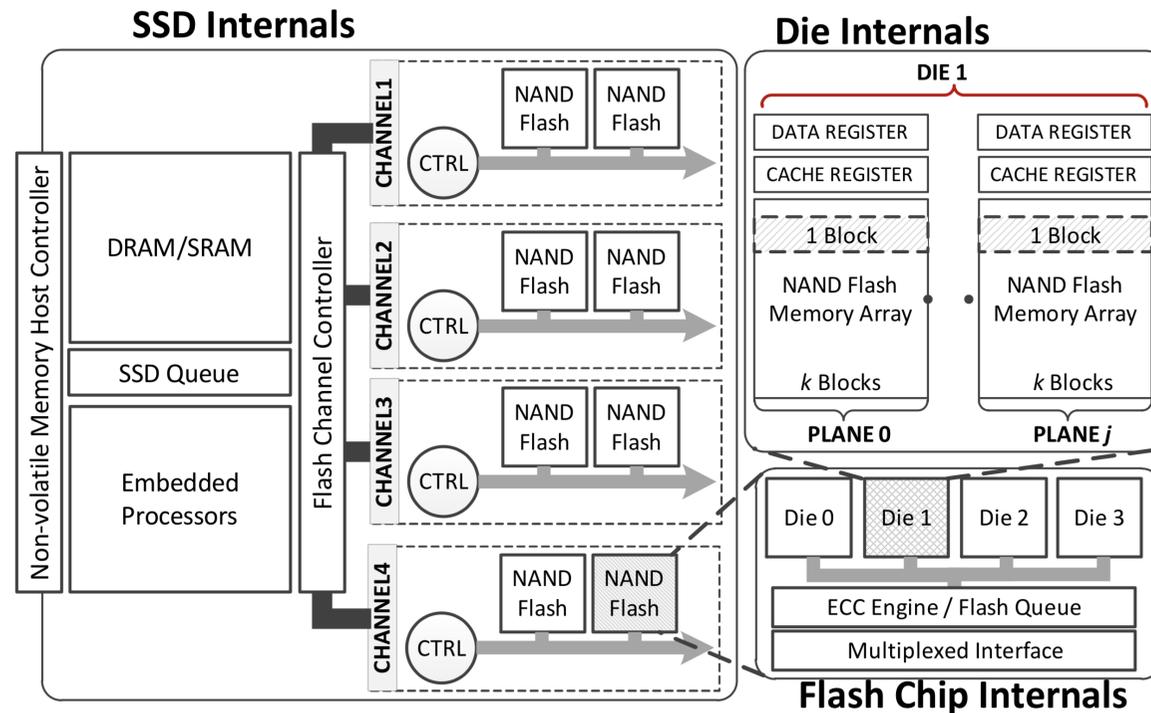
- Daten werden auf **Seiten** gespeichert (4kB-16kB pro Seite)
- Lesen einer Seite ist sehr schnell, Richtwert  $100 \mu s$
- Seiten werden in sogenannte **Erase-Blocks** zusammengefasst, die aus 128-256 Seiten bestehen.
- Bevor eine Seite beschrieben werden kann, muss sie gelöscht werden (kein in-place Update).
- Löschen einzelner Seiten ist jedoch nicht möglich, sondern der gesamte Erase-Block muss gelöscht werden.
- Löschen ist teuer:  $2000-5000 \mu s$

# SSD - Schreibvorgang

- **Schreibvorgang:** logische Seite X schreiben
  - logische Seite X ist intern auf eine physische Seite Y abgebildet
  - schreiben auf Seite X: statt alte physische Seite Y zu löschen wird Schreibbefehl auf freie physische Seite Z umgelenkt
  - die Abbildung zwischen logischen und physischen Seiten wird aktualisiert: X wird nun auf Z abgebildet
  - physische Seite Y wird als “löschar” markiert
- **Garbage Collection:**
  - Erase-Block mit vielen “löscharen” Seiten wird als Ganzes gelöscht
  - erfordert gegebenenfalls noch nicht gelöschte Seiten vorher zu kopieren
- **Wear Leveling**
  - nur begrenzte Anzahl von Löschar-Operationen möglich: 100k-1M
  - deshalb wird versucht, Erase-Blocks gleichmäßig zu verwenden

# SSD - Performance in der Praxis/1

- Theoretisch sollten SSDs:
  - für random und sequentiellen Zugriff gleich schnell sein
  - lesen sollte schneller sein als schreiben
- In der Praxis sind SSDs hoch komplex aufgebaut:



Jung, Kandemir. Revisiting widely held SSD expectations and rethinking system-level implications. SIGMETRICS'13

# SSD - Performance in der Praxis/2

- Performance hängt von verschiedenen **SSD-internen Effekten** ab:
  - Garbage Collection, Wear Leveling
  - Interne Parallelität auf mehreren Ebenen: channels, flash chips, dies
  - Flash-Speicher wird fehleranfälliger, wenn er öfters gelesen oder geschrieben wird
  - Leseoperationen führen zu Schreiboperationen, wenn der gelesene Bereich nur mehr schwer lesbar ist
- **Black-Box Verhalten:**
  - Sequentielles Lesen bis 2x schneller als nicht-sequentielles (random)
  - Zugriffszeit verdoppelt sich in einem gealterten SSD Laufwerk
  - Schreibgeschwindigkeit ist zwischen random und sequentiellem Lesen
  - random Schreiben verlangsamt das spätere Lesen
  - Write Cliff: im schlimmsten Fall kann Schreiben (random) sehr langsam werden (langsamer als Festplatte)

# SSD - Fazit

- SSDs sind im Schnitt deutlich schneller als Festplatten
- SSDs sind auch deutlich teurer
- random access (Lesen oder Schreiben) soll sowohl bei SSDs als auch bei Festplatten vermieden werden

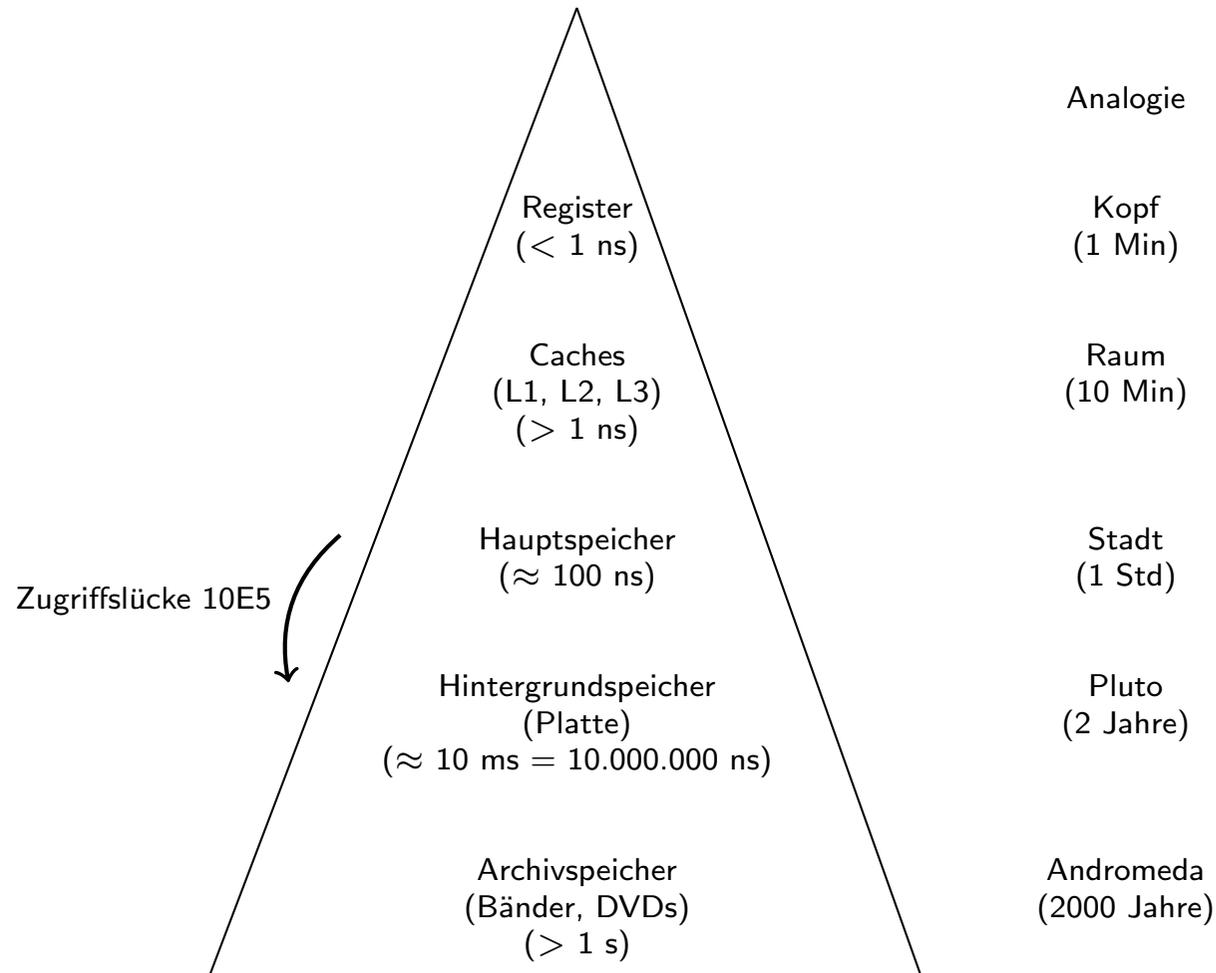
# Zusammenfassung

- flüchtige und persistente Speicher
- Speicherpyramide
  - Primärspeicher: Cache, RAM
  - Sekundärspeicher: SSDs, Festplatten
  - Tertiärspeicher: Bänder, optische Platten
- SSDs können Festplatten in manchen Bereichen ersetzen
- Persistenter RAM verspricht interessante Entwicklungen

# Inhalt

- 1 Speichermedien
- 2 Speicherzugriff
- 3 Dateien, Datensätze, Blöcke

# Speicherhierarchie



# Platten Zugriff Optimieren

- Wichtiges Ziel von DBMSs: Transfer von Daten zwischen Platten und Hauptspeicher möglichst effizient gestalten.
  - optimieren/minimieren der Anzahl der Zugriffe
  - minimieren der Anzahl der gelesenen/geschriebenen Blöcke
  - so viel Blöcke als möglich im Hauptspeicher halten (→ Puffer Manager)
- Techniken zur Optimierung des Block Speicher Zugriffs:
  1. Disk Arm Scheduling
  2. Geeignete Dateistrukturen
  3. Schreib-Puffer und Log Disk

# Block Speicher Zugriff/1

- **Disk Arm Scheduling:** Zugriffe so ordnen, dass Bewegung des Arms minimiert wird.
- **Elevator Algorithm (Aufzug-Algorithmus):**
  - Disk Controller ordnet die Anfragen nach Spur (von innen nach außen oder umgekehrt)
  - Bewege Arm in eine Richtung und erledige alle Zugriffe unterwegs bis keine Zugriffe mehr in diese Richtung vorhanden sind
  - Richtung umkehren und die letzten beiden Schritte wiederholen

# Block Speicher Zugriff/2

- **Datei Organization:** Daten so in Blöcken speichern, wie sie später zugegriffen werden.
  - z.B. verwandte Informationen auf benachbarten Blöcken speichern
- **Fragmentierung:** Blöcke einer Datei sind nicht hintereinander auf der Platte abgespeichert
  - Gründe für Fragmentierung sind z.B.
    - Daten werden eingefügt oder gelöscht
    - die freien Blöcke auf der Platte sind verstreut, d.h., auch neue Dateien sind schon zerstückelt
  - sequentieller Zugriff auf fragmentierte Dateien erfordert erhöhte Bewegung des Zugriffsarms
  - manche Systeme erlauben das Defragmentieren des Dateisystems

# Block Speicher Zugriff/3

Schreibzugriffe können asynchron erfolgen um Throughput (Zugriffe/Sekunde) zu erhöhen

- **Persistente Puffer:** Block wird zunächst auf persistenten RAM (RAM mit Batterie-Backup oder Flash Speicher) geschrieben; der Controller schreibt auf die Platte, wenn diese gerade nicht beschäftigt ist oder der Block zu lange im Puffer war.
  - auch bei Stromausfall sind Daten sicher
  - Schreibzugriffe können geordnet werden um Bewegung des Zugriffsarms zu minimieren
  - Datenbank Operationen, die auf sicheres Schreiben warten müssen, können fortgesetzt werden
- **Log Disk:** Eine Platte, auf die der Log aller Schreibzugriffe sequentiell geschrieben wird
  - wird gleich verwendet wie persistenter RAM
  - Log schreiben ist sehr schnell, da kaum Spurwechsel erforderlich
  - erfordert keine spezielle Hardware

# Puffer Manager/1

- **Puffer:** Hauptspeicher-Bereich für Kopien von Platten-Blöcken
- **Puffer Manager:** Subsystem zur Verwaltung des Puffers
  - Anzahl der Platten-Zugriffe soll minimiert werden
  - ähnlich der virtuellen Speicherverwaltung in Betriebssystemen

# Puffer Manager/2

- Programm fragt Puffer Manager an, wenn es einen Block von der Platte braucht.
- Puffer Manager **Algorithmus**:
  1. Programm fordert Plattenblock an.
  2. Falls Block nicht im Puffer ist:
    - Der Puffer Manager reserviert Speicher im Puffer (wobei nötigenfalls andere Blöcke aus dem Puffer geworfen werden).
    - Ein rausgeworfener Block wird nur auf die Platte geschrieben, falls er seit dem letzten Schreiben auf die Platte geändert wurde.
    - Der Puffer Manager liest den Block von der Platte in den Puffer.
  3. Der Puffer Manager gibt dem anfordernden Programm die Hauptspeicheradresse des Blocks im Puffer zurück.
- Es gibt verschiedene Strategien zum Ersetzen von Blöcken im Puffer.

# Ersetzstrategien für Pufferseiten/1

- **LRU Strategie** (least recently used): Ersetze Block der am längsten nicht benutzt wurde.
  - Idee: Zugriffsmuster der Vergangenheit benutzen um zukünftiges Verhalten vorherzusagen
  - erfolgreich in Betriebssystemen eingesetzt
- **MRU Strategie**: (most recently used): Ersetze zuletzt benutzten Block als erstes.
  - LRU kann schlecht für bestimmte Zugriffsmuster in Datenbanken sein, z.B. wiederholtes Scannen von Daten
- **Pinned block**: Darf nicht aus dem Puffer entfernt werden.
- **Toss Immediate Strategy**: Block wird sofort rausgeworfen, wenn das letzte Tupel bearbeitet wurde.

# Ersetzstrategien für Pufferseiten/2

- Anfragen in DBMSs haben wohldefinierte Zugriffsmuster (z.B. sequentielles Lesen) und das DBMS kann die Information aus den Benutzeranfragen verwenden, um zukünftig benötigte Blöcke vorherzusagen.
- **Beispiel:** Berechne  $R \bowtie S$  mit Nested Loops:
  - für jedes Tupel  $tr$  von  $R$ :
    - für jedes Tupel  $ts$  von  $S$ :
      - wenn  $ts$  und  $tr$  das Join-Prädikat erfüllen, dann ...
- Verschiedene Zugriffsmuster für  $R$  und  $S$ 
  - ein  $R$ -Block wird nicht mehr benötigt, sobald das letzte Tupel des Blocks bearbeitet wurde; er sollte also sofort entfernt werden, auch wenn er gerade erst benutzt worden ist
  - ein  $S$ -Block wird nochmal benötigt, wenn alle anderen  $S$ -Blöcke abgearbeitet sind

# Ersetzstrategien für Pufferseiten/3

## Informationen für Ersatzstrategien in DBMSs:

- Zugriffspfade haben **wohldefinierte Zugriffsmuster** (z.B. sequentielles Lesen)
- **Information im Anfrageplan** um zukünftige Blockanfragen vorherzusagen
- **Statistik** über die Wahrscheinlichkeit, dass eine Anfrage für eine bestimmte Relation kommt
  - z.B. das Datenbankverzeichnis (speichert Schema) wird oft zugegriffen
  - Heuristik: Verzeichnis im Hauptspeicher halten

# Inhalt

- 1 Speichermedien
- 2 Speicherzugriff
- 3 Dateien, Datensätze, Blöcke**

# Datei Organisation

- **Datei:** (file) aus logischer Sicht eine Reihe von Datensätzen
  - ein *Datensatz* (record) ist eine Reihe von Datenfeldern
  - mehrere Datensätze in einem Platten-Block
  - *Kopfteil* (header): Informationen über Datei (z.B. interne Organisation)
- **Abbildung von Datenbank in Dateien:**
  - eine Relation wird in eine Datei gespeichert
  - ein Tupel entspricht einem Datensatz in der Datei
- **Cooked vs. raw files:**
  - cooked: DBMS verwendet Dateisystem des Betriebssystems (einfacher, code reuse)
  - raw: DBMS verwaltet Plattenbereich selbst (unabhängig von Betriebssystem, bessere Performance, z.B. Oracle)
- **Fixe vs. variable Größe von Datensätzen:**
  - fix: einfach, unflexibel, Speicher-ineffizient
  - variabel: komplizierter, flexibel, Speicher-effizient

# Fixe Datensatzlänge/1

- **Speicheradresse:**  $i$ -ter Datensatz wird ab Byte  $m * (i - 1)$  gespeichert, wobei  $m$  die Größe des Datensatzes ist
- mehrere Möglichkeiten zum **Löschen des  $i$ -ten Datensatzes:**

(a) verschiebe Datensätze  $i + 1, \dots, n$  nach  $i, \dots, n - 1$

(b) verschiebe letzten Datensatz nach  $i$

(c) nicht verschieben, sondern "Free List" verwalten

record 0	A-102	Perryridge	400
record 1	A-305	Round Hill	350
record 2	A-215	Mianus	700
record 3	A-101	Downtown	500
record 4	A-222	Redwood	700
record 5	A-201	Perryridge	900
record 6	A-217	Brighton	750
record 7	A-110	Downtown	600
record 8	A-218	Perryridge	700

# Fixe Datensatzlänge/2

- **Free List:**
  - speichere Adresse des ersten freien Datensatzes im Kopfteil der Datei
  - freier Datensatz speichert Pointer zum nächsten freien Datensatz
- der Speicherbereich des gelöschten Datensatzes wird für Free List Pointer verwendet
- **Beispiel:** Free List nach löschen der Datensätze 4, 6, 1 (in dieser Reihenfolge)

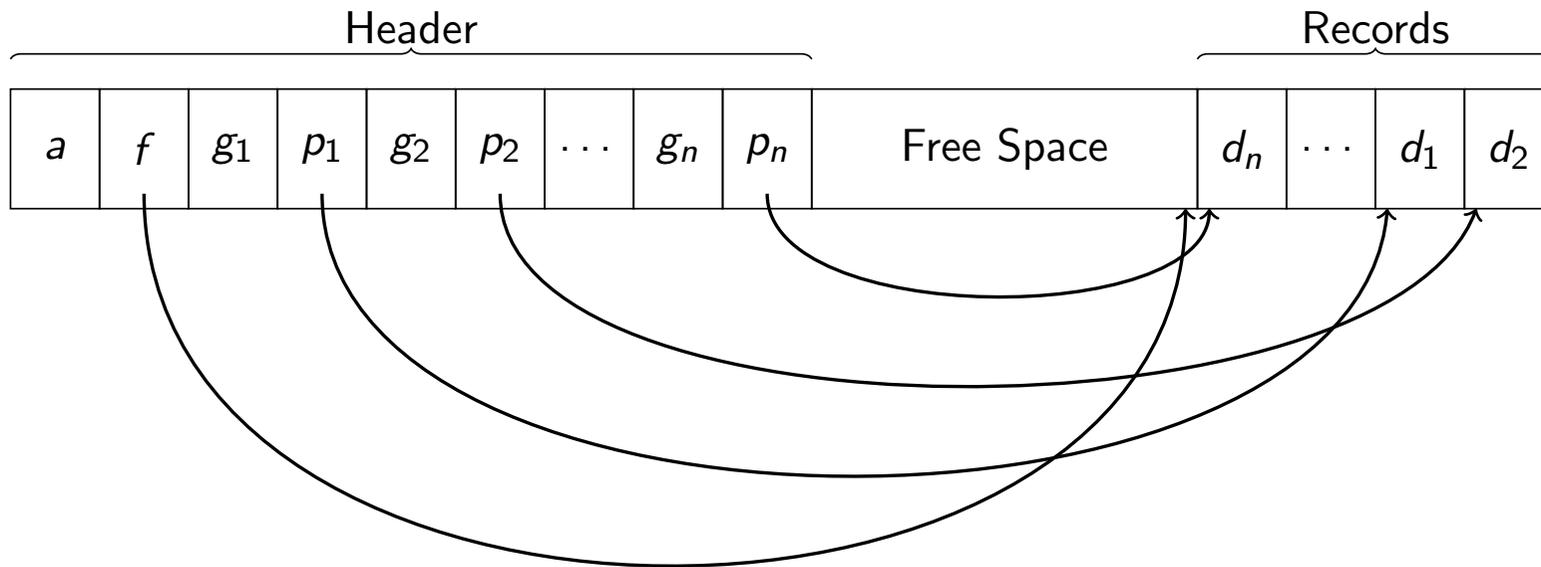
header			
record 0	A-102	Perryridge	400
record 1			
record 2	A-215	Mianus	700
record 3	A-101	Downtown	500
record 4			
record 5	A-201	Perryridge	900
record 6			
record 7	A-110	Downtown	600
record 8	A-218	Perryridge	700

# Variable Datensatzlänge/1

- Warum Datensätze mit variabler Größe?
  - Datenfelder variabler Länge (z.B., VARCHAR)
  - verschiedene Typen von Datensätzen in einer Datei
  - Platz sparen: z.B. in Tabellen mit vielen null-Werten (häufig in der Praxis)
- Datensätze verschieben kann erforderlich werden:
  - Datensätze können größer werden und im vorgesehenen Speicherbereich nicht mehr Platz haben
  - neue Datensätze werden zwischen existierenden Datensätzen eingefügt
  - Datensätze werden gelöscht (leere Zwischenräume verhindern)
- Pointer soll sich nicht ändern:
  - alle existierenden Referenzen zum Datensatz müssten geändert werden
  - das wäre kompliziert und teuer
- Lösung: Slotted Pages (TID-Konzept)

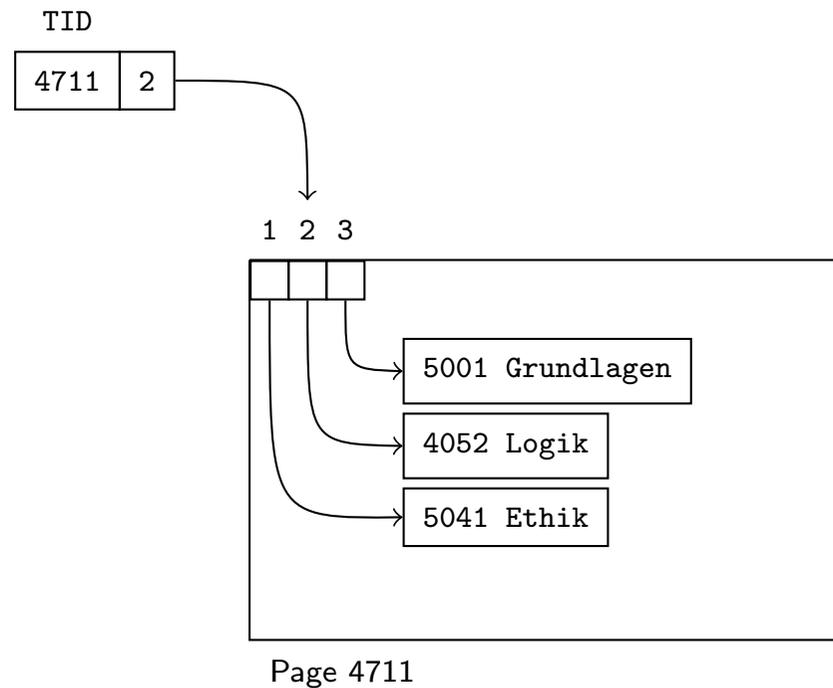
# Slotted Pages/1

- Slotted Page:
  - Kopfteil (header)
  - freier Speicher (free space)
  - Datensätze (records)  $d_1, d_2, \dots, d_n$
- Kopfteil speichert:
  - Anzahl der Datensätze  $a$
  - Ende des freien Speichers  $f$
  - Größe  $g_i$  und Pointer  $p_i$  auf Startposition jedes Datensatzes  $d_i$



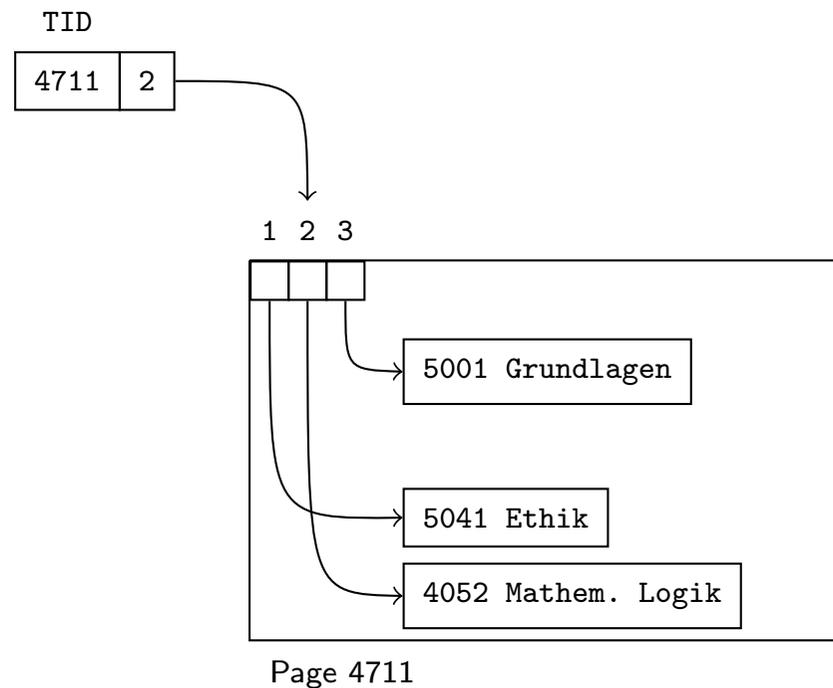
# Slotted Pages/2

- **TID**: Tuple Identifier besteht aus
  - Nummer des Blocks (page ID)
  - Offset des Pointers zum Datensatz
- Datensätze werden **nicht direkt adressiert**, sondern über TID



# Slotted Pages/3

- Verschieben innerhalb des Blocks:
  - Pointer im Kopfteil wird geändert
  - TID ändert sich nicht



# Slotted Pages/4

- Verschieben zwischen Blöcken:
  - Datensatz wird ersetzt durch Referenz auf neuen Block, welche nur intern genutzt wird
  - Zugriff auf Datensatz erfordert das Lesen von zwei Blöcken
  - TID des Datensatzes ändert sich nicht
  - weitere Verschiebungen modifizieren stets die Referenz im ursprünglichen Block (d.h. es entsteht keine verkettete Liste)

